CLEAN CODE:

SOURCE CODE STRUCTURE

Presentors: Hanie Solaty Nia, Sara Soltani, Fargol Shirvani Far

SUMMARY OF CONTENTS

(OUR MAIN TOPICS TODAY)

- What is Clean Code?
- What's The Advantages?
- Why Do We Care About Clean Code?
- Rules and Tips
- Source Code Structure/Formatting
- Conclusion
- Resources

WHAT IS A CLEAN CODE?

(DEFINITION)

- Clean code is code that is easy to understand and easy to change.
- “understandable” means that the code can be immediately understood by any experienced developer
- Easy to undestand:

1. the execution flow of the entire application
2. how the different objects collaborate with each other
3. the role and responsibility of each class
4. what each method does
5. what is the purpose of each expression and variable

WHAT IS A CLEAN CODE? (2)

(DEFINITION)

- Code is easy to modify if it can be adapted and extended.
- Easy to change means:

1. Classes and methods are small and only have single responsibility
2. Classes have clear and concise public APIs
3. Classes and methods are predictable and work as expected
4. The code is easily testable and has unit tests (or it is easy to write the tests)
5. Tests are easy to understand and easy to change.

Advantages

INDEPENDENT

Clean code does not depend on the original developer and any programmer can work with the code

EASIER TO MAINTAIN

This prevents problems that may occur when working with legacy code, for example.

EASIER TO DEBUG

The code is easy to extend and refactor, and is easy to fix bugs in the codebase.

IMPORTANCE OF CLEAN CODE

- WORKING EFFICIENTLY ON CODE
- REDUCING THE COST OF MAINTENANCE OF THE APPLICATION
- MAKING IT EASIER TO ESTIMATE THE TIME NEEDED FOR NEW FEATURES.
- MAKING IT EASIER TO FIX BUGS

AND...

RULES AND TIPS

1- GENERAL RULES

2- DESIGN RULES

3- UNDERSTANDABILITY TIPS

4- NAMES RULES

5- FUNCTIONS RULES

6- COMMENTS RULES

7- SOURCE CODE STRUCTURE

8- OBJECTS AND DATA STRUCTURES

9- TESTS

10- CODE SMELLS

SOURCE CODE STRUCTURE

SOURCE CODE STRUCTURE - 1

SEPARATE CONCEPTS VERTICALLY

Bad Example:

```
class Foo {metho1(){}metho2(){}}
class Bar {metho1(){}metho2(){}}
```

Good Example:

```
class Foo {
  method1() {}

  method2() {}
}

class Bar {
  method1() {}

  method2() {}
}
```

SOURCE CODE STRUCTURE - 2

RELATED CODE SHOULD APPEAR VERTICALLY DENSE

- We can group variable declarations together without blank lines and classes in their own group as follows:

```
let x = 1;
let y = 2;

class Foo {
  method1() {}

  method2() {}
}
```

SOURCE CODE STRUCTURE - 3

DECLARE VARIABLES CLOSE TO THEIR USAGE.

- Variables should be declared as similar as possible to the spot they are used.
- Because our functions are very short, local variables should appear at the top of each function.
- Control variables for loops should usually be declared within the loop statement.

SOURCE CODE STRUCTURE - 4

DEPENDENT FUNCTIONS SHOULD BE CLOSE.

- Dependent functions should be grouped together.
- This helps you to read the code very quickly without navigating too far around various places within the code.

```
class Foo {
  foo() {
    this.bar();
  }

  bar() {
    this.baz();
  }

  baz() {}
}
```

SOURCE CODE STRUCTURE - 5

SIMILAR FUNCTIONS SHOULD BE CLOSE.

- Related code should be near each other so that we don’t have to scroll to find related concepts in our code.
- The more affinity is greater, the less vertical space between them can be.

```
class Assert {
  assertTrue() {}

  assertFalse() {}

  assertNotUndefined() {}
}
```

SOURCE CODE STRUCTURE - 6

PLACE FUNCTIONS IN THE DOWNWARD DIRECTION.

- The function that’s called should be below a function that does the calling.
- We expect the most relevant ideas first, and the low specifics are supposed to finish last.
- This creates a nice flow from high to a low level.

```
class Foo {
  foo() {
    this.bar();
  }

  bar() {
    this.baz();
  }

  baz() {}
}
```

SOURCE CODE STRUCTURE - 7

KEEP LINES SHORT.

- We will have horizontal lines that are wider than before since windows are larger than in previous days.
- The norm for the former days was 80 characters long, but now 100–120 is okay.
- The argument is that most people do not scroll to read our code horizontally.

SOURCE CODE STRUCTURE - 8

DON'T USE HORIZONTAL ALIGNMENT.

Bad Example:

```
let num     = 1;
let value   = 2;
```

Good Example:

```
let num = 1;
let value = 2;
```

SOURCE CODE STRUCTURE - 9

USE WHITE SPACE TO ASSOCIATE RELATED THINGS AND DISASSOCIATE WEAKLY RELATED.

```
class Calculator {
  constructor(a, b) {
    this.a = a;
    this.b = b;
  }

  add() {
    return this.a + this.b;
  }
```

```
subtract() {
  return this.a - this.b;
}

multiply() {
  return this.a * this.b;
}

divide() {
  return this.a / this.b;
}
```

SOURCE CODE STRUCTURE - 10

DON'T BREAK INDENTATION.

Bad Example:

```
const loop = ()=>{if(true){for(let x of [1,2,3]){console.log(x)}}};
```

SOURCE CODE STRUCTURE - 10

DON'T BREAK INDENTATION.

Good Example:

```
const loop = () => {
  if (true) {
    for (let x of [1, 2, 3]) {
      console.log(x)
    }
  }
};
```

CONCLUSION

- Code formatting can increase the readability and interpretation of a code.
- Don’t waste your time manually formatting your code.
- Take advantage of the amazing modern tools out there : Prettier, TSLint, CodeMetrics

RESCOURCES

[1]:https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29#source-code-structure

[2]:https://medium.com/nerd-for-tech/clean-code-formatting-source-code-structure-f3021575d79

[3]:https://garywoodfine.com/what-is-clean-code/

[4]: https://www.ionos.com/digitalguide/websites/web-development/clean-code-principles-advantages-and-examples/

[5]: https://cvuorinen.net/2014/04/what-is-clean-code-and-why-should-you-care/

[6]: https://www.codegrip.tech/productivity/everything-you-need-to-know-about-code-smells/

[7]: https://mosquitolwz.medium.com/clean-code-quotes-5-formatting-b7bdd4a828d6#:~:text=Concepts%20that%20are%20closely%20related,have%20a%20very%20good%20reason.

[8]: https://levelup.gitconnected.com/javascript-clean-code-vertical-formatting-d15097f5f30e

“

Thank You!

”